CertiK Verification Report For ContentOS

Request Date: 2019-04-15
Revision Date: 2019-04-28

Disclaimer

This Report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Verification Services Agreement between CertiK and ContentOS(the "Company"), or the scope of services/verification, and terms and conditions provided to the Company in connection with the verification (collectively, the "Agreement"). This Report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This Report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

PASS

CERTIK *believes this smart contract passes security qualifications to be listed on digital asset exchanges.*

Apr 28, 2019

Summary

This audit report summarises the smart contract verification service requested by ContentOS. The goal of this security audit is to guarantee that the audited smart contracts are robust enough to avoid any potential security loopholes.
The result of this report is only a reflection of the source code that was determined in this scope, and of the source code at the time of the audit.

Type of Issues

CertiK smart label engine applied 100% coveraged formal verification labels on the source code, and scanned the code using our proprietary static analysis and formal verification engine to detect the follow type of issues.

Title	Description	Issues	SWC ID
Integer Overflow and Underflow	An overflow/underflow happens when an arithmetic operation reaches the maximum or minimum size of a type.	0	SWC-101
Function incorrectness	Function implementation does not meet the specification, leading to intentional or unintentional vulnerabilities.	0	
Buffer Overflow	An attacker is able to write to arbitrary storage locations of a contract if array of out bound happens	0	SWC-124
Reentrancy	A malicious contract can call back into the calling contract before the first invocation of the function is finished.	0	SWC-107
Transaction Order Dependence	A race condition vulnerability occurs when code depends on the order of the transactions submitted to it.	0	SWC-114
Timestamp Dependence	Timestamp can be influenced by minors to some degree.	4	SWC-116

Insecure Compiler Version	Using an fixed outdated compiler version or floating pragma can be problematic, if there are publicly disclosed bugs and issues that affect the current compiler version used.	2	SWC-102 SWC-103
Insecure Randomness	Block attributes are insecure to generate random numbers, as they can be influenced by minors to some degree.	0	SWC-120
"tx.origin" for authorization	tx.origin should not be used for authorization. Use msg.sender instead.	0	SWC-115
Delegatecall to Untrusted Callee	Calling into untrusted contracts is very dangerous, the target and arguments provided must be sanitized.	0	SWC-112
State Variable Default Visibility	Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.	0	SWC-108
Function Default Visibility	Functions are public by default. A malicious user is able to make unauthorized or unintended state changes if a developer forgot to set the visibility.	0	SWC-100
Uninitialized variables	Uninitialized local storage variables can point to other unexpected storage variables in the contract.	0	SWC-109
Assertion Failure	The assert() function is meant to assert invariants. Properly functioning code should never reach a failing assert statement.	0	SWC-110
Deprecated Solidity Features	Several functions and operators in Solidity are deprecated and should not be used as best practice.	0	SWC-111
Unused variables	Unused variables reduce code quality	0	

Review Details

Source Code SHA-256 Checksum

- **lock_linear.sol** `4225963563907cc509e5827fcc329073cdc6b9cefba3da713bc3e7d1540e59a6`
- **lock_nonlinear.sol** `f09b94f509d104f171fd74761db812da1ffb58e759692259861c9da0401888f3`
- **lock_reward.sol** ** `1a5d7eeac6454710623a0929ae6b0867672f11ca84e4f698f6d479d546049132`

Summary

CertiK team is invited by The ContentOS team to audit the design and implementations of its TimeLock and Vesting Contracts, and the source code has been analyzed under different perspectives and with different tools such as CertiK formal verification checkings as well as manual reviews by smart contract experts. We have been actively interacting with client-side engineers when there was any potential loopholes or recommended design changes during the audit process, and ContentOS team has been actively giving us updates for the source code and feedback about the business logics.

We suggest more unit test cases to be added into the repository to simulate different `release` token scenarios to those `benefinaries`. i.e: release in the linear fashion- constant amount && release in non-linear fashion - cliff vesting.

Meanwhile, it is recommended to have a more well-detailed document for the public to describe the source code specifications and implementations.

Overall we found the ContentOS team is very professional, provide with well-detailed document for the public to describe the source code specifications and implementations. The contracts follow good practices, with reasonable amount of features on top of the ERC20 related to administrive controls by the token issuer. With the final update of source code and delivery of the audit report, we conclude that the contract is not vulnerable to any classically known antipatterns or security issues. The audit report itself is not necessarily a guarantee of correctness or trustworthiness, and we always recommend seeking multiple opinions, more test coverage and sandbox deployments before the mainnet release.

Recommandation

Items in this section are low impact to the overall aspects of the smart contracts, thus will let client to decide whether to have those reflected in the final deployed version of source codes.

lock_linear.sol

- `isAuthorized()` – can be simplified to `return owner == src;`
- `set_lock_info()`
  - `set_lock_info()` can be modified many time before the `lock()` is called, `dateArray.push(tmp * days_of_month)` keep dataArray increasing, instead start from index 0.
- `set_benificiary()` – Ensure the address `b` is not zero contract
- `require()` – after version 0.4.22, `require()` allow to optional message for giving more context to the user about what type of condition is not match i.e: `require(_to != address(0), "invalid, _to is a zero address")`

lock_nonlinear.sol

- `isAuthorized()` – can be simplified to

```
return src == owner
;
```

- `release_specific()` –
  - **line 111:** `require(false);` is not given enough information about the condition is not meet to the user. Adding messages definitely could help for futher investigation purpose. i.e. `require(false, "index must be between 1 - 4");`

For every issues found, CertiK categorizes them into 3 buckets based on its risk level:

- Critical: The code implementation does not match the specification, or it could result in loss of funds for contract owner or users.
- Medium: The code implementation does not match the specification at certain condition, or it could affect the security standard by lost of access control.
- Low: The code implementation is not a best practice, or use a suboptimal design pattern, which may lead to security vulnerability, but no concern found yet.

Source Code with CertiK Labels

File lock_nonlinear.sol

```
1  pragma solidity ^0.4.24;
2
3      contract DAO {
4          function balanceOf(address addr) public returns (uint);
5      }
6
7      interface RegisterInterface {
8          function register(string);
9      }
10
11 // auth
12 contract Auth {
13     address     public owner;
14     /*@CTK Auth
15       @post __post.owner == msg.sender
16      */
17     constructor () public {
18          owner = msg.sender;
19     }
20
21     modifier auth {
22         require(isAuthorized(msg.sender) == true);
23         _;
24     }
25
26     /*@CTK isAuthorized
27       @post __return == (src == owner)
28      */
29     function isAuthorized(address src) internal view returns (bool) {
30         if(src == owner){
31             return true;
32         } else {
33             return false;
34         }
35     }
36 }
37
38     contract TokenTimelock is Auth{
39
40     constructor() public {
41         benificiary = msg.sender;
42     }
43
44     uint constant public days_of_month = 30;
45
46     uint  public firstTime = 0;
47     uint  public secondTIme = 0;
48     uint  public thirdTime = 0;
49     uint  public fourthTime = 0;
50     mapping (uint => bool) public release_map;
51
52     uint256 public totalFutureRelease = 0;
53
54
```

```
55      // cosToken address,
56      address constant public contract_addr = 0x589891a198195061cb8ad1a75357a3b7dbadd7bc
            ;
57
58      address public benificiary;
59
60      uint    public startTime;
61      bool public lockStart = false;
62
63      // set total cos to lock
64      /*@CTK set_total
65        @tag assume_completion
66        @post !lockStart
67        @post msg.sender == owner
68        @post __post.totalFutureRelease == total
69       */
70      function set_total(uint256 total) auth public {
71          require(lockStart == false);
72          totalFutureRelease = total;
73      }
74
75      // set month to release
76      /*CTK set_release_month
77        @tag no_overflow
78        @post __post.firstTime == months1 * days_of_month
79        @post __post.secondTIme == months2 * days_of_month
80        @post __post.thirdTime == months3 * days_of_month
81        @post __post.fourthTime == months4 * days_of_month
82       */
83      function set_release_month(int months1,int months2,int months3,int months4) auth
            public {
84          require(lockStart == false);
85          require(months1 > 0);
86          require(months2 > 0);
87          require(months3 > 0);
88          require(months4 > 0);
89          firstTime = uint(months1) * days_of_month;
90          secondTIme = uint(months2) * days_of_month;
91          thirdTime = uint(months3) * days_of_month;
92          fourthTime = uint(months4) * days_of_month;
93
94          require(firstTime < secondTIme );
95          require(secondTIme < thirdTime);
96          require(thirdTime < fourthTime);
97      }
98
99      // when transfer certain balance to this contract address, we can call lock
100     /*@CTK lock
101       @tag assume_completion
102       @post !lockStart
103       @post firstTime != 0
104       @post secondTIme != 0
105       @post thirdTime != 0
106       @post fourthTime != 0
107       @post __post.startTime == block.timestamp + offsetMinutes * 60
108       @post __post.lockStart
109      */
110     // function lock(int offsetMinutes) auth public returns(bool) {
```

```
111     function lock(uint offsetMinutes) auth public returns(bool) {
112         require(lockStart == false);
113         require(firstTime != 0);
114         require(secondTIme != 0);
115         require(thirdTime != 0);
116         require(fourthTime != 0);
117         require(offsetMinutes >= 0);
118 
119         DAO cosTokenApi = DAO(contract_addr);
120         uint256 balance = cosTokenApi.balanceOf(address(this));
121         require(balance == totalFutureRelease);
122 
123         startTime = block.timestamp + uint(offsetMinutes) * 1 minutes;
124         lockStart = true;
125     }
126 
127     /*@CTK set_benificiary
128       @tag assume_completion
129       @post msg.sender == owner
130       @post __post.benificiary == b
131      */
132     function set_benificiary(address b) auth public {
133         benificiary = b;
134     }
135 
136     function release_specific(uint index,uint i) private {
137         if (release_map[i] == true) {
138             emit mapCheck(true,i);
139             return;
140         }
141         emit mapCheck(false,i);
142 
143         DAO cosTokenApi = DAO(contract_addr);
144         uint256 balance = cosTokenApi.balanceOf(address(this));
145         uint256 eachRelease = 0;
146         if (index == 1) {
147             eachRelease = totalFutureRelease / 10;
148         } else if (index >= 2 && index <= 4) {
149             eachRelease = (totalFutureRelease / 10) * 3;
150         } else {
151             require(false);
152         }
153 
154         bool ok = balance >= eachRelease;
155         emit balanceCheck(ok,balance);
156         require(balance >= eachRelease);
157 
158         bool success = contract_addr.call(bytes4(keccak256("transfer(address,uint256)")
                ),benificiary,eachRelease);
159         emit tokenTransfer(success);
160         require(success);
161         release_map[i] = true;
162     }
163 
164     event mapCheck(bool ok,uint window);
165     event balanceCheck(bool ok,uint256 balance);
166     event tokenTransfer(bool success);
167 
```

```
168     function release() auth public {
169         require(lockStart == true);
170         require(release_map[fourthTime] == false);
171         uint theDay = dayFor();
172         // release day must be after lock day
173         require(theDay > firstTime);
174 
175         if ( theDay > firstTime && theDay <= secondTIme) {
176             release_specific(1,firstTime);
177         } else if (theDay > secondTIme && theDay <= thirdTime) {
178             release_specific(1,firstTime);
179             release_specific(2,secondTIme);
180         } else if (theDay > thirdTime && theDay <= fourthTime) {
181             release_specific(1,firstTime);
182             release_specific(2,secondTIme);
183             release_specific(3,thirdTime);
184         } else if (theDay > fourthTime) {
185             release_specific(1,firstTime);
186             release_specific(2,secondTIme);
187             release_specific(3,thirdTime);
188             release_specific(4,fourthTime);
189         }
190     }
191 
192         // days after lock
193     /*@CTK dayFor
194       @post block.timestamp < startTime -> __return == 0
195       @post block.timestamp >= startTime -> __return == (block.timestamp - startTime)
              / 86400 + 1
196      */
197     function dayFor() view public returns (uint) {
198         uint timestamp = block.timestamp;
199         return timestamp < startTime ? 0 : (timestamp - startTime) / 1 days + 1;
200     }
201 
202     function regist(string key) auth public {
203         RegisterInterface(contract_addr).register(key);
204     }
205 }
```

File lock_linear.sol

```
1  pragma solidity ^0.4.24;
2 
3      contract DAO {
4          /*@CTK balanceOf
5            @tag spec
6            @post __return == 5
7           */
8          function balanceOf(address addr) public returns (uint);
9      }
10 
11     interface RegisterInterface {
12         function register(string);
13     }
14 
15 // auth
16 contract Auth {
17     address     public owner;
```

```
18      /*@CTK Auth
19        @post __post.owner == msg.sender
20       */
21      constructor () public {
22          owner = msg.sender;
23      }
24
25      modifier auth {
26          require(isAuthorized(msg.sender) == true);
27          _;
28      }
29
30      /*@CTK isAuthorized
31        @post __return == (src == owner)
32       */
33      function isAuthorized(address src) internal view returns (bool) {
34          if(src == owner){
35              return true;
36          } else {
37              return false;
38          }
39      }
40  }
41
42  contract TokenTimelock is Auth{
43      /*@CTK TokenTimeLock
44        @post __post.benificiary == msg.sender
45       */
46      constructor() public {
47          benificiary = msg.sender;
48      }
49
50      uint constant public days_of_month = 30;
51
52      uint[] public dateArray;
53      uint public release_percent = 0;
54
55      mapping (uint => bool) public release_map;
56      uint256 public totalFutureRelease = 0;
57
58      // cosToken address,
59      address constant public contract_addr = 0x589891a198195061cb8ad1a75357a3b7dbadd7bc
            ;
60      address public benificiary;
61      uint    public startTime;
62      bool public lockStart = false;
63
64      // set total cos to lock
65      /*@CTK set_total
66        @tag assume_completion
67        @post !lockStart
68        @post msg.sender == owner
69        @post __post.totalFutureRelease == total
70       */
71      function set_total(uint256 total) auth public {
72          require(lockStart == false);
73          totalFutureRelease = total;
74      }
```

```
75
76      // set month to release
77      function set_lock_info(int startMonth,int periods,int percent,int gap) auth public
            {
78          require(lockStart == false);
79          require(startMonth > 0);
80          require(periods > 0);
81          require(percent > 0);
82          require(gap > 0);
83          require(periods * percent == 100);
84          release_percent = uint(percent);
85          uint tmp = uint(startMonth);
86          delete dateArray;
87          /*CTK ForLoop_set_lock_info
88            @inv dateArray[dateArray.length - 1] == tmp__pre * days_of_month
89            @inv tmp == tmp__pre + gap
90            @inv i < periods
91            @inv gap == gap__pre
92            @inv periods == periods__pre
93            @post i == periods
94            @post !__should_return
95           */
96          for (int i = 0; i < periods; i++) {
97              dateArray.push(tmp * days_of_month);
98              tmp += uint(gap);
99          }
100     }
101
102     // when transfer certain balance to this contract address, we can call lock
103     function lock(int offsetMinutes) auth public returns(bool) {
104         require(lockStart == false);
105         require(offsetMinutes >= 0);
106         for(uint i = 0; i < dateArray.length; i++) {
107             require(dateArray[i] != 0);
108         }
109         require(release_percent != 0);
110         require(totalFutureRelease != 0);
111
112         DAO cosTokenApi = DAO(contract_addr);
113         uint256 balance = cosTokenApi.balanceOf(address(this));
114         require(balance == totalFutureRelease);
115
116         startTime = block.timestamp + uint(offsetMinutes) * 1 minutes;
117         lockStart = true;
118     }
119
120     /*@CTK set_benificiary
121       @tag assume_completion
122       @post msg.sender == owner
123       @post __post.benificiary == b
124      */
125     function set_benificiary(address b) auth public {
126         benificiary = b;
127     }
128
129     function release_specific(uint i) private {
130         if (release_map[i] == true) {
131             emit mapCheck(true,i);
```

```
132             return;
133         }
134         emit mapCheck(false,i);
135 
136         DAO cosTokenApi = DAO(contract_addr);
137         uint256 balance = cosTokenApi.balanceOf(address(this));
138         uint256 eachRelease = 0;
139         eachRelease = (totalFutureRelease / 100) * release_percent;
140 
141         bool ok = balance >= eachRelease;
142         emit balanceCheck(ok,balance);
143         require(balance >= eachRelease);
144 
145         bool success = contract_addr.call(bytes4(keccak256("transfer(address,uint256)")
                ),benificiary,eachRelease);
146         emit tokenTransfer(success);
147         require(success);
148         release_map[i] = true;
149     }
150 
151     event mapCheck(bool ok,uint window);
152     event balanceCheck(bool ok,uint256 balance);
153     event tokenTransfer(bool success);
154 
155     function release() auth public {
156         require(lockStart == true);
157         require(release_map[dateArray[dateArray.length-1]] == false);
158         uint theDay = dayFor();
159 
160         for (uint i=0; i<dateArray.length;i++) {
161             if(theDay > dateArray[i]) {
162                 release_specific(dateArray[i]);
163             }
164         }
165     }
166 
167         // days after lock
168     /*@CTK dayFor
169       @post block.timestamp < startTime -> __return == 0
170       @post block.timestamp >= startTime -> __return == (block.timestamp - startTime)
              / 86400 + 1
171      */
172     function dayFor() view public returns (uint) {
173         uint timestamp = block.timestamp;
174         return timestamp < startTime ? 0 : (timestamp - startTime) / 1 days + 1;
175     }
176 
177     function regist(string key) auth public {
178         RegisterInterface(contract_addr).register(key);
179     }
180 }
```

How to read

Detail for Request 1

transferFrom to same address

Verification date	20, Oct 2018
Verification timespan	395.38 ms

CERTIK *label location*	Line 30-34 in File howtoread.sol

CERTIK *label*

```
30    /*@CTK FAIL "transferFrom to same address"
31        @tag assume_completion
32        @pre from == to
33        @post __post.allowed[from][msg.sender] ==
34    */
```

Raw code location	Line 35-41 in File howtoread.sol

Raw code

```
35    function transferFrom(address from, address to
          ) {
36        balances[from] = balances[from].sub(tokens
37        allowed[from][msg.sender] = allowed[from][
38        balances[to] = balances[to].add(tokens);
39        emit Transfer(from, to, tokens);
40        return true;
41    }
```

Counterexample	This code violates the specification

Initial environment

```
1  Counter Example:
2  Before Execution:
3      Input = {
4          from = 0x0
5          to = 0x0
6          tokens = 0x6c
7      }
8      This = 0
                    }
53                  balance: 0x0
54              }
55      }
56
```

Post environment

```
57 After Execution:
58     Input = {
59         from = 0x0
60         to = 0x0
61         tokens = 0x6c
```

Static Analysis Request

INSECURE_COMPILER_VERSION

Line 1 in File lock_nonlinear.sol

```
1 pragma solidity ^0.4.24;
```

Only these compiler versions are safe to compile your code: 0.4.25

TIMESTAMP_DEPENDENCY

Line 124 in File lock_nonlinear.sol

```
124         startTime = block.timestamp + uint(offsetMinutes) * 1 minutes;
```

"block.timestamp" can be influenced by minors to some degree

TIMESTAMP_DEPENDENCY

Line 199 in File lock_nonlinear.sol

```
199         uint timestamp = block.timestamp;
```

"block.timestamp" can be influenced by minors to some degree

INSECURE_COMPILER_VERSION

Line 1 in File lock_linear.sol

```
1 pragma solidity ^0.4.24;
```

Only these compiler versions are safe to compile your code: 0.4.25

TIMESTAMP_DEPENDENCY

Line 116 in File lock_linear.sol

```
116         startTime = block.timestamp + uint(offsetMinutes) * 1 minutes;
```

"block.timestamp" can be influenced by minors to some degree

TIMESTAMP_DEPENDENCY

Line 173 in File lock_linear.sol

```
173         uint timestamp = block.timestamp;
```

"block.timestamp" can be influenced by minors to some degree

Formal Verification Request 1

Auth

28, Apr 2019
6.6 ms

Line 14-16 in File lock_nonlinear.sol

```
14    /*@CTK Auth
15      @post __post.owner == msg.sender
16     */
```

Line 17-19 in File lock_nonlinear.sol

```
17    constructor () public {
18        owner = msg.sender;
19    }
```

The code meets the specification

Formal Verification Request 2

isAuthorized

28, Apr 2019
13.13 ms

Line 26-28 in File lock_nonlinear.sol

```
26    /*@CTK isAuthorized
27      @post __return == (src == owner)
28     */
```

Line 29-35 in File lock_nonlinear.sol

```
29    function isAuthorized(address src) internal view returns (bool) {
30        if(src == owner){
31            return true;
32        } else {
33            return false;
34        }
35    }
```

The code meets the specification

Formal Verification Request 3

set_total

28, Apr 2019
62.3 ms

Line 64-69 in File lock_nonlinear.sol

```
64    /*@CTK set_total
65      @tag assume_completion
66      @post !lockStart
67      @post msg.sender == owner
68      @post __post.totalFutureRelease == total
69     */
```

Line 70-73 in File lock_nonlinear.sol

```
70    function set_total(uint256 total) auth public {
71        require(lockStart == false);
72        totalFutureRelease = total;
73    }
```

The code meets the specification

Formal Verification Request 4

lock

28, Apr 2019
114.69 ms

Line 100-109 in File lock_nonlinear.sol

```
100   /*@CTK lock
101     @tag assume_completion
102     @post !lockStart
103     @post firstTime != 0
104     @post secondTIme != 0
105     @post thirdTime != 0
106     @post fourthTime != 0
107     @post __post.startTime == block.timestamp + offsetMinutes * 60
108     @post __post.lockStart
109    */
```

Line 111-126 in File lock_nonlinear.sol

```
111   function lock(uint offsetMinutes) auth public returns(bool) {
112       require(lockStart == false);
113       require(firstTime != 0);
114       require(secondTIme != 0);
115       require(thirdTime != 0);
116       require(fourthTime != 0);
117       require(offsetMinutes >= 0);
118
119       DAO cosTokenApi = DAO(contract_addr);
120       uint256 balance = cosTokenApi.balanceOf(address(this));
121       require(balance == totalFutureRelease);
122
123       startTime = block.timestamp + uint(offsetMinutes) * 1 minutes;
124       lockStart = true;
125   }
```

The code meets the specification

Formal Verification Request 5

set_benificiary

28, Apr 2019
37.26 ms

Line 128-132 in File lock_nonlinear.sol

```
128   /*@CTK set_benificiary
129     @tag assume_completion
130     @post msg.sender == owner
131     @post __post.benificiary == b
132    */
```

Line 133-135 in File lock_nonlinear.sol

```
133   function set_benificiary(address b) auth public {
134       benificiary = b;
135   }
```

The code meets the specification

Formal Verification Request 6

dayFor

28, Apr 2019
12.6 ms

Line 194-197 in File lock_nonlinear.sol

```
194   /*@CTK dayFor
195     @post block.timestamp < startTime -> __return == 0
196     @post block.timestamp >= startTime -> __return == (block.timestamp - startTime)
          / 86400 + 1
197    */
```

Line 198-201 in File lock_nonlinear.sol

```
198   function dayFor() view public returns (uint) {
199       uint timestamp = block.timestamp;
200       return timestamp < startTime ? 0 : (timestamp - startTime) / 1 days + 1;
201   }
```

The code meets the specification

Formal Verification Request 7

Auth

28, Apr 2019
6.6 ms

Line 18-20 in File lock_linear.sol

```
18    /*@CTK Auth
19      @post __post.owner == msg.sender
20     */
```

Line 21-23 in File lock_linear.sol

```
21    constructor () public {
22        owner = msg.sender;
23    }
```

✅ The code meets the specification

Formal Verification Request 8

isAuthorized

📅 28, Apr 2019
⏱ 13.13 ms

Line 30-32 in File lock_linear.sol

```
30    /*@CTK isAuthorized
31      @post __return == (src == owner)
32     */
```

Line 33-39 in File lock_linear.sol

```
33    function isAuthorized(address src) internal view returns (bool) {
34        if(src == owner){
35            return true;
36        } else {
37            return false;
38        }
39    }
```

✅ The code meets the specification

Formal Verification Request 9

TokenTimeLock

📅 28, Apr 2019
⏱ 8.79 ms

Line 43-45 in File lock_linear.sol

```
43    /*@CTK TokenTimeLock
44      @post __post.benificiary == msg.sender
45     */
```

Line 46-48 in File lock_linear.sol

```
46    constructor() public {
47        benificiary = msg.sender;
48    }
```

✅ The code meets the specification

Formal Verification Request 10

set_total

28, Apr 2019
65.4 ms

Line 65-70 in File lock_linear.sol

```
65    /*@CTK set_total
66      @tag assume_completion
67      @post !lockStart
68      @post msg.sender == owner
69      @post __post.totalFutureRelease == total
70     */
```

Line 71-74 in File lock_linear.sol

```
71    function set_total(uint256 total) auth public {
72        require(lockStart == false);
73        totalFutureRelease = total;
74    }
```

The code meets the specification

Formal Verification Request 11

set_benificiary

28, Apr 2019
37.96 ms

Line 120-124 in File lock_linear.sol

```
120   /*@CTK set_benificiary
121     @tag assume_completion
122     @post msg.sender == owner
123     @post __post.benificiary == b
124    */
```

Line 125-127 in File lock_linear.sol

```
125   function set_benificiary(address b) auth public {
126       benificiary = b;
127   }
```

The code meets the specification

Formal Verification Request 12

dayFor

28, Apr 2019
12.72 ms

Line 168-171 in File lock_linear.sol

```
168     /*@CTK dayFor
169       @post block.timestamp < startTime -> __return == 0
170       @post block.timestamp >= startTime -> __return == (block.timestamp - startTime)
              / 86400 + 1
171      */
```

Line 172-175 in File lock_linear.sol

```
172     function dayFor() view public returns (uint) {
173         uint timestamp = block.timestamp;
174         return timestamp < startTime ? 0 : (timestamp - startTime) / 1 days + 1;
175     }
```

The code meets the specification